On Tree Decomposability of Henneberg Graphs

Marta R. Hidalgo, Robert Joan-Arinyo
Grup d'Informàtica a l'Enginyeria
Universitat Politècnica de Catalunya
Barcelona, Catalonia

Abstract

In this work we describe an algorithm that generates well constrained geometric constraint graphs which are solvable by the tree-decomposition constructive technique. The algorithm is based on Henneberg constructions and would be of help in transforming under-constrained problems into well constrained problems as well as in exploring alternative constructions over a given set of geometric elements.

1 Introduction

A fundamental issue in parametric geometric modeling is to find descriptions of how to place with respect to each other a set of given basic geometric objects in such a way that a set of constraints defined on them hold. Once a description has been found the aim is to explore different actual placements, and possibly different sizes, by changing values assigned to some parameters on which the description depends. Examples of questions related to the parametric geometric modeling are: i) How can we find the parametric description? ii) Once the description has been found, does this description allow to generate all possible placements of the geometric components? Conversely, iii) given a placement for the geometric objects, can we check whether the constraints defined on them actually hold? Answering questions i) and ii) is the goal of geometric constraint solving. Question iii) entails a geometric analysis problem which can be solved by measuring and checking.

Many techniques have been reported in the literature that provide powerful and efficient methods for solving geometric problems defined by constraints. For a review, see Hoffmann *et al.*, [8]. Computer programs that solve geometric problems defined by constraints are called *solvers*. Among all the geometric constraint solving techniques, our interest here focuses on the one

known as *constructive*. See [9, 10, 12, 13, 15] and the references there in for an in depth discussion on this topic.

Constructive solvers belong to the DR-solvers class [10] and have two main components: the *analyzer* and the *constructor*. Given the geometric elements and the constraints defined on them, the analyzer figures out a description of how geometric elements are placed with respect to each other in such a way that the constraints are fulfilled. This description is called *construction plan*.

If the analyzer succeeds, actual values are assigned to the parameters and the constructor builds an instance of a placement for the geometric objects, provided that no numerical incompatibility arises due to geometric degeneracy.

In general, for the analyzer to succeed the problem must be well constrained, that is, the number of solutions to the problem is finite. If a number of constraints are missing, the geometric realization is no longer rigid and there are an infinity of solutions to the problem. This scenario can be found, for example, in the first steps in parametric solid modeling when design requirements are not well defined yet. Some times the situation arises from the fact that the designer is only interested in fixing constraints for features which play a central role in the design. An efective way of limiting the number of solutions to the problem is transforming it into a well constrained one [16, 17].

Devising a method to automatically define sets of constraints over a given set of geometric elements would be a tool for the designers to explore alternative solutions when developing new products. However, for this tool to be useful and effective, the resulting parametric geometric constraint problem must be well constrained and solvable.

With the aim of contributing to improve previously developed solutions to the issues considered above, in this work we describe an approach to automatically generate a set of constraints that defines a well constrained problem over a set of geometric elements. The approach is based on Henneberg constructions and guarantees that the constraint problem is tree-decomposable and therefore solvable by constructive graph-based DR-solvers whenever the starting problem is abstracted as the triangle graph K_3. However, the approach clearly applies as far as the starting graph with missing constraints is tree-decomposable.

2 Preliminaries

We consider geometric constraint problems in the Euclidian plane. The geometric elements are points, straight line segments, circles and arcs of circle with constant radii. The geometric constraints defined on these geometric elements include distance between two points, perpendicular distance between a point and a segment and angle between two stright line segments. Incidence, perpendicularity, parallelism, tangency and concentricity constraints can also be defined. These constraints can always be represented in terms of distance and angle constraints, [20]. Under these assumptions, a geometric constraint problem can be represented by a *geometric constraint graph*, $G = (V, E)$, where the nodes $V(G)$ are the geometric elements and the constraints are the graph edges, $E(G)$.

2.1 Graph Rigidity Characterization

Each segment or point in a geometric constraint problem has two degrees of freedom. Each distance or angle constraint corresponds to one equation and therefore cancels one degree of freedom. Intuitively, if the total number of degrees of freedom equals the number of constraints and each degree of freedom is canceled by a different constraint in the given set, we expect that all the geometric elements in the problem will be placed with respect to each other in such a way that the constraints are satisfied. Note that the resulting geometric object will be a rigid body with three remaining degress of freedom, two translations and one rotation. In these conditions, the geometric constraint problem has a finite number of solutions for non-degenerate configurations and we say that the geometric constraint problem is well defined or well constrained or generically rigid.

Graphs such that abstract geometric objects which are invariant under rigid transformations of translation and rotation are called *rigid*. Different kinds of rigidity have been defined, such as minimal or global rigidity, see [21, 11]. According to Laman, [19], the rigidity of a geometric constraint problem can be characterized through the notion of generic rigidity of the associated geometric constraint graph as follows

Theorem 2.1 A graph $G = (V, E)$ in the Euclidian plane is rigid if and only if there is a subset E' of E such that:

1. $|E'| = 2|V| - 3$,
2. $|F| \leq |V(F)| - 3$ for all non-empty subsets F of E'.

Condition 1 is usually interpreted as requiring that $E(G)$ contain enough

edges to be rigid. Condition 2, known as the *Laman's condition*, is interpreted as requiring that none of the subsets of $V(G)$ packs too many edges.

Strictly speaking, characterization of generic rigidity by Laman's condition is limited to problems realizable as a set of bars and joints, that is, geometric problems build from points and distances. However, Laman's condition allows to solve a sizeble amount of practical problems including also lines as geometric elements and angles and tangencies as geometric constraints.

A complet set of definitions related to the concept of rigidity can be found in [7, 14] and [23]. Technically, the notion of well constrained graph can be formalized as follows, [4].

Definition 2.1 Let $G = (V, E)$ be a geometric constraint graph.

1. G is *over-constrained* if there is an induced subgraph with $m \leq |V|$ vertices and more than $2m - 3$ edges.
2. G is *under-constrained* if it is not over-constrained and $|E| < 2|V| - 3$.
3. G is *well constrained* if it is not over-constrained and $|E| = 2|V| - 3$.

In this work we only consider well constrained geometric constraint problems which can be abstracted as Laman graphs.

2.2 Tree Decomposable Graphs

One of the tools most widely used to build graph-based DR-solvers is the *tree decomposition* also known as *triangular decomposition*. In this approach, the geometric constraint problem is abstracted as a well constrained graph and the construction plan yielded by the analyzer is the tree decomposition of the graph, if one has been found.

We are given a graph $G = (V, E)$ where V is a finite set of nodes which stand for the geometric objects in the constraint problem, and E is a collection of edges, where each edge is a geometric constraint defined on two geometric elements. An edge is an unordered pair (u, v) of distinct vertices $u, v \in V(G)$. In general $V(G)$ and $E(G)$ will denote respectively the set of vertices and edges of the graph G.

Consider the graph $G = (V, E)$ and let $G_1(V_1, E_1)$, $G_2(V_2, E_2)$ and $G_3(V_3, E_3)$ be three subgraphs of G such that

$$V = V_1 \cup V_2 \cup V_3, \qquad E = E_1 \cup E_2 \cup E_3$$

the set of vertices pairwise share one vertex

$$V_1 \cap V_2 = \{a\}, \quad V_2 \cap V_3 = \{b\}, \quad V_3 \cap V_1 = \{c\}$$

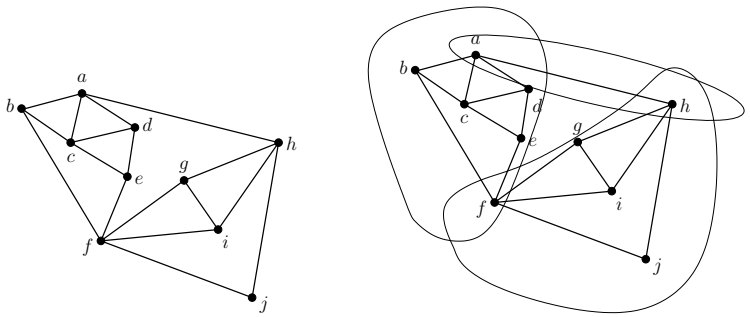

Figure 1: a) Graph. b) Graph tree decomposition step.

and the set of edges are pairwise disjoint

$$E_1 \cap E_2 = \emptyset, \quad E_2 \cap E_3 = \emptyset, \quad E_3 \cap E_1 = \emptyset$$

We say that G_1, G_2 and G_3 are a tree decomposition step of G for which $\{a, b, c\}$ is a *triple of hinges*. Consider the graph in Figure 1a. The set of vertices $\{a, h, f\} \in V(G)$ is a triple of hinges which induces the decomposition step shown in Figure 1b. Each subgraph G_i, $1 \leq i \leq 3$, induces a well defined geometric constraint subproblem called *cluster*.

Let $G = (V, E)$ be a graph. We say that a ternary tree T is a *tree decomposition* of G if

1. G is the root of T,
2. Each node $G' \subset G$ of T is the father of exactly three nodes, say G'_1, G'_2 and G'_3, which are the clusters output by a tree decomposition step applied to the subgraph G', and
3. Each leaf node contains a cluster with exactly two vertices $\{a, b\}$ of $V(G)$ such that edge (a, b) is in $E(G)$.

A graph for which there is a tree decomposition is called *tree decomposable*. In general, a tree decomposition of a graph is not unique. Figure 2 shows two different tree decompositions for the graph given in Figure 1a. For the sake of clarity, tree decompositions only show the set of vertices included by clusters. Labels on the tree edges include triples of hinges and will be defined later on.

We close this setion listing a few observations concerning Laman graphs.

Observation 2.2 Laman graphs have no disconnecting points and no vertices with degree zero nor one.

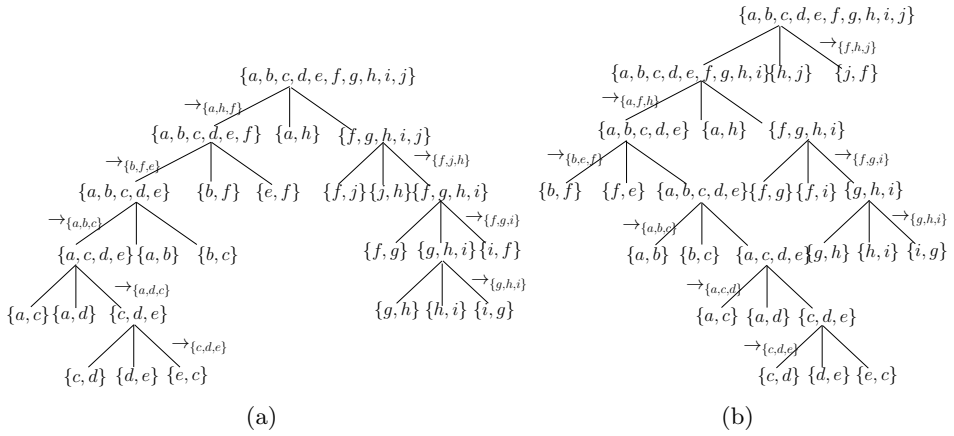

Figure 2: Two different tree decompositions for the graph shown in Figure 1a.

Observation 2.3 Every vertex $u \in V(G)$ in a tree-decomposable graph $G = (V, E)$ belongs to at least one triple $\{u, v, w\}$ which induces a tree decomposition step in G.

Observation 2.4 Vertices $u, v \in V(G)$ in a tree-decomposable graph $G = (V, E)$ bounding an edge $(u, v) \in E(G)$ belong to at least one triple $\{u, v, w\}$ which induces a tree decomposition step in G.

Observation 2.5 Let $G = (V, E)$ be a graph and $G_1 = (V_1, E_1), G_2 = (V_2, E_2), G_3 = (V_3, E_3)$ be a tree decomposition step of G. If there is a vertex $v \in V(G_i)$ for some G_i in $\{G_1, G_2, G_3\}$ with degree 1 then $|V(G_i)| = 2$ and $|E(G_i)| = 1$, that is $G_i = (\{v_{i1}, v_{i2}\}, \{(v_{i1}, v_{i2})\})$.

2.3 The Tree Decomposition as a Rewrite System

As shown in [3, 13], the process of actually building a solution to a geometric constraint problem described as a tree decomposition of a graph can be abstracted as a rewrite system, [18], where terms are sets of clusters. Given a graph $G = (V, E)$ the starting set of clusters is defined as

$$\mathbf{C}_G = \{\{u, v\} : (u, v) \in E(G)\}$$

Clusters are rewritten using a tree decomposition step as reduction rule denoted $\rightarrow$ and formally defined as follows.

Definition 2.2 Let $\mathbf{C}$ be a set of clusters where there are three clusters $C_i, 1 \leq i \leq 3$ such that pairwise share one vertex $C_1 \cap C_2 = \{a\}, C_2 \cap C_3 = \{b\}, C_3 \cap C_1 = \{c\}$ with a, b and c distinct. Then $\mathbf{C} \rightarrow \mathbf{C}^*$ where

$$\mathbf{C}^* = (\mathbf{C} - \{C_1, C_2, C_3\}) \cup \{C_1 \cup C_2 \cup C_3\}$$

Definition 2.3 A *derivation* is a sequence of applications of the rewriting rule in $(\mathbf{C}, \rightarrow)$. We will denote a derivation by

$$\mathbf{C} \rightarrow^* \mathbf{C}^*$$

.

Definition 2.4 A term $\mathbf{C}^*$ is derived from $\mathbf{C}$ if and only if there is a derivation such that $\mathbf{C} \rightarrow^* \mathbf{C}^*$. A term $\mathbf{C}$ to which the tree decomposition rule does not apply is called *irreducible* or *normal form.*

The reduction system $(\mathbf{C}_G, \rightarrow)$ has a unique normal form that is obtained after finitely many reductions, [3, 13]. In this conditions, if the geometric constraint problem is well constrained, that is, the problem has a finite number of solution instances, the derivation reduces the initial set $\mathbf{C}_G$ to a single cluster. The sequence of construction steps identified by the derivation places a fixed set of triples of geometric elements in relative positions such that the constraints hold.

If $\{a, b, c\}$ are the nodes pairwise shared by clusters C_1, C_2, C_3, denote by $\rightarrow_{\{a,b,c\}}$ the reduction that merges the clusters. In this conditions, the set

$$\mathbf{T_C}(*) = \{\{a, b, c\} \ : \rightarrow_{\{a,b,c\}} \in \rightarrow^*\}$$

is the set of hinges triples in the derivation $\rightarrow^*$.

According to [3] and [6], each triple of hinges is used once and only once in a reduction process. This means that given two different reduction sequences over the same starting and ending terms, $\mathbf{C} \rightarrow^* \mathbf{C}^*$ and $\mathbf{C} \rightarrow^{*'} \mathbf{C}^*$ we have

$$\mathbf{T_C}(*) = \mathbf{T_C}(*')$$

Edges in the tree decompositions shown in Figure 2, are labeled with the reduction that merges three clusters into a new one. The set of hinges triples is the same in both tree decompositions,

$$\mathbf{T_C}(*) = \{\{a, b, c\}, \{a, d, c\}, \{a, h, f\}, \{b, f, e\}, \{c, d, e\}, \{f, i, g\}, \{f, i, j\}, \{g, h, i\}\}$$

For a more formal treatement of this concepts see [4] and [13].

3 Henneberg Constructions

In this section we recall Henneberg constructions in the plane. First we recall the basic construction rules. Then we formalize Henneberg sequences as a rewrite system. For an indepth study on Henneberg constructions see [5, 22, 24].

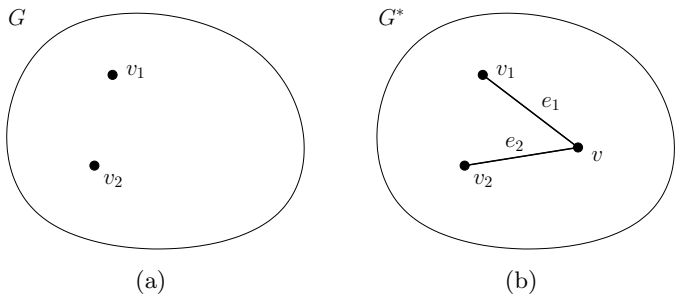

Figure 3: a) Graph. b) Graph G^* derived from gaph G by application of a Henneberg I construction.

3.1 Basic Henneberg Constructions

Henneberg constructions include two different construction steps defined as follows, [5].

1. *Henneberg I step.* Let $G = (V, E)$ be a graph with two distinct vertices $v_1, v_2 \in V(G)$ and $G^* = (V^*, E^*)$ be the graph obtained by attaching to G a new vertex v with edges (v, v_1) and (v, v_2). Then G^* is the graph derived from G by a Henneberg I step. See Figure 3.

2. *Henneberg II step.* Let $G = (V, E)$ be a graph with an edge $e = (v_1, v_2) \in E(G)$ and a third vertex $v_3 \in V(G)$. The graph $G^* = (V^*, E^*)$ obtained from G by deleting the edge (v_1, v_2) and inserting a new vertex v plus three edges $(v, v_1), (v, v_2)$ and (v, v_3) is the graph derived from G by a Henneberg II step. See Figure 4.

In what follows Henneberg I and Henneberg II step constructions will be denoted as H1S and H2S respectively.

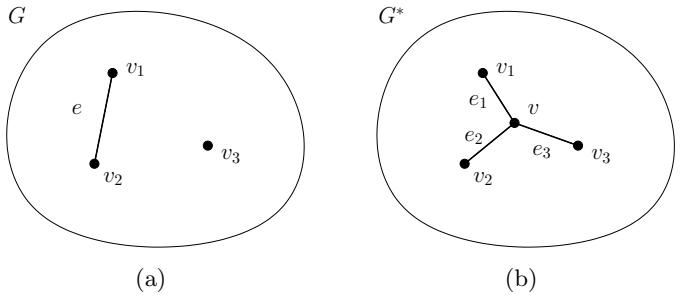

Figure 4: a) Graph G. b) Graph G^* derived from graph G by the application of a Henneberg II construction.

3.2 Henneberg Sequences

Let $\mathbf{G}$ be the set of graphs and let $\rightarrow_1$ and $\rightarrow_2$ be respectively the H1S and the H2S step constructions and let $\rightarrow_{\mathbf{HS}}$ be de set including both construction steps. Then the pair $(\mathbf{G}, \rightarrow_{\mathbf{HS}})$ is a rewrite system the starting term of which is the graph $\mathbf{G}$ with reductions $\rightarrow_{\mathbf{HS}}$ [18]. Now, however, the rewrite system is not terminating and there are no normal forms.

A *derivation* in $(\mathbf{G}, \rightarrow_{\mathbf{HS}})$ is any sequence

$$G_0 \rightarrow_{\mathbf{HS}} G_1 \rightarrow_{\mathbf{HS}} G_2 \rightarrow_{\mathbf{HS}} \ldots \rightarrow_{\mathbf{HS}} G_k$$

of applications of rewrite rules in $\rightarrow_{\mathbf{HS}}$. G_0 is the starting term.

In general a derivation in $(\mathbf{G}, \rightarrow_{\mathbf{HS}})$ will be written as $G_0 \rightarrow^*_{\mathbf{HS}} G^*$. When the derivation includes just one of the two rules in $\rightarrow_{\mathbf{HS}}$, it will be denoted as either $G_0 \rightarrow^*_1 G^*$ or $G_0 \rightarrow^*_2 G^*$. We will refer to the specific derivation where the starting graph is the triangle K_3, denoted $K_3 \rightarrow^*_{\mathbf{HS}} G$, as the *Henneberg derivation* of G.

We are specifically interested in two different families of graphs generated by a Henneberg derivation [1, 2]. The Henneberg I family, denoted $\mathbf{H}_1$, includes those graphs G derived by $K_3 \rightarrow^*_1 G$. The Henneberg II family, denoted $\mathbf{H}_2$, includes those graphs G derived by $K_3 \rightarrow^*_{\mathbf{HS}} G$. As shown in [5, 24], $\mathbf{H}_2$ and the set of Laman graphs are the same set.

3.3 H1S and Tree-decomposability

It is easy to see that the H1S derivation preserves tree-decomposability thus the following result holds.

Theorem 3.1 Let G and G^* be two graphs such that $G \rightarrow_1 G^*$. Then G^* is tree-decomposable if and only if G is tree-decomposable.

Proofs for results in this section have been included in Appendix A. They can also be found in [6]. Figure 5 shows a sequence of H1S constructions that yields a tree-decomposable graph. Just consider as tree decomposition the Henneberg sequence reversed.

3.4 H2S and Tree-decomposability

Henneberg sequences which include H2S constructions create well constrained rigid graphs, however these graphs are not necessarily tree-decomposable. Figure 6 shows a Henneberg sequence for the Desargues graph, [1], where the H2S construction removes edge (a, d) and adds the new vertex f plus edges $(f, a), (f, d)$ and (f, e). The resulting graph is well constrained and rigid but is not tree-decomposable.

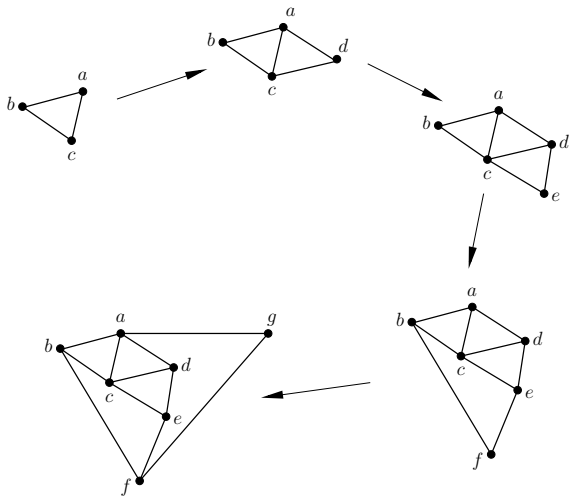

Figure 5: A sequence of H1S constructions guarantees that the graph generated is tree-decomposable.

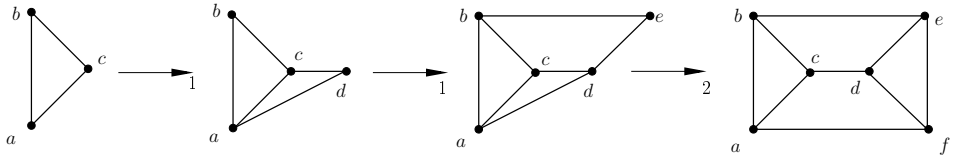

Figure 6: A Henneberg sequence in H2S that creates the Desargues graph which is well constrained but is not tree-decomposable.

Next we establish the conditions under which an H2S construction preserves graph tree-decomposability.

Theorem 3.2 Let $G = (V, E)$ be a graph in the $\mathbf{H}_1$ family and G^* be the graph such that $G \rightarrow_2 G^*$, where the H2S construction involves the edge $(v_1, v_2) \in E(G)$ and vertex $v_3 \in V(G)$. If the triple $\{v_1, v_2, v_3\}$ are hinges for a reduction in the derivation $K_3 \rightarrow_1^* G$, then $G^* \in \mathbf{H}_1$.

Therefore, graphs created by derivations $K_3 \rightarrow_{\mathbf{HS}}^* G$ are tree-decomposable whenever reductions $\rightarrow_2$ are applied under the conditions fixed in Theorem 3.2.

Lemma 3.3 Let $G = (V, E)$ be a tree-decomposable Henneberg graph for which G_1, G_2, G_3 is a decomposition induced by the triple $\{v_1, v_2, v_3\}$. Let $G_i = (\{a, b\}, \{(a, b)\})$ for some $1 \leq i \leq 3$ with $a, b \in \{v_1, v_2\}$ and $a \neq b$. Then the graph G^* created by the reduction $G \rightarrow_2 G^*$ involving the edge $(a, b) \in E(G)$ and vertex $v_3 \in V(G)$ is tree-decomposable.

Lemma 3.4 Let $G = (V, E)$ be a tree-decomposable Henneberg graph for which G_1, G_2, G_3 is a decomposition induced by the triple $\{v_1, v_2, x\}$ and such that edge $(v_1, v_2) \in E(G_i)$ for some $1 \leq i \leq 3$. If there is a vertex $w \in V(G_i)$ such that the triple $\{v_1, v_2, w\}$ decomposes G_i, then the graph G^* created by the reduction $G \rightarrow_2 G^*$ involving edge $(v_1, v_2) \in E(G)$ and a third vertex $v_3 \in V(G)$ is tree-decomposable.

For the final result we need to recall from graph theory the concept of *lowest common ancestor*. Let $\mathbf{T}$ be a rooted tree. A node $u \in \mathbf{T}$ is an *ancestor* of a node $v \in \mathbf{T}$ if the path from the root of $\mathbf{T}$ to v goes through u. A node $w \in \mathbf{T}$ is a *common ancestor* of u and v if it is an ancestor of both u and v. The *lowest common ancestor* of nodes $u, v \in \mathbf{T}$ is the common ancestor of nodes u, v for which the path from the root is maximum.

Now, with each tree-decomposable graph $G = (V, E)$ we associate a rooted tree $\mathbf{T}$ corresponding to the tree-decomposition of G. Notice that each node in $\mathbf{T}$ is a subgraph of G and that the root, $\mathbf{T}_0$, is the given graph G.

Finally, we define the *lowest common ancestor of vertices* $u, v, w, \ldots \in V(G)$ in $\mathbf{T}$ as the lowest common ancestor of those leaf nodes in $\mathbf{T}$ which include vertices $u, v, w, \ldots \in V(G)$. In what follows we shall denote the lowest common ancestor of vertices in the tree-decomposition $\mathbf{T}$ as $LCA_T(u, v, w, \ldots)$. We do not allow a node to be descendant of itself.

Theorem 3.5 Let $G = (V, E)$ be a tree-decomposable Henneberg graph with $\mathbf{T}$ the associated tree-decomposition. Let G^* be the grah created by the derivation $K_3 \rightarrow_{\mathbf{HS}} G' \rightarrow_{\mathbf{HS}} G \rightarrow_2 G^*$ where reduction $\rightarrow_2$ is applied on edge $(v_1, v_2) \in E(G') \subset E(G)$ and vertex $v_3 \in V(G') \subset V(G)$. Let $\hat{G}_1, \hat{G}_2, \hat{G}_3$ be a decomposition of $LCA_T(v_1, v_2, v_3)$ with $(v_1, v_2) \in E(\hat{G}_i)$ for some $1 \leq i \leq 3$. Then G^* is tree-decomposable if either $\hat{G}_i = (\{v_1, v_2\}, \{(v_1, v_2)\})$

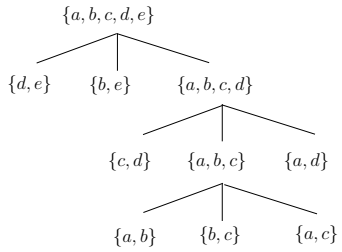

Figure 7: Tree decomposition for the graph created in Figure 7 after the second H1S reduction, $\rightarrow_1$.

or there is a vertex $w \in V(\hat{G}_i)$ such that $\{v_1, v_2, w\}$ is a triple of hinges for $\hat{G}_i$.

To illustrate Theorem 3.5 consider the Henneberg sequence in Figure 6 that creates the Desargues graph. Figure 7 shows a tree decomposition, $\mathbf{T}$, for the graph created after the second H1S reduction, $\rightarrow_1$. The H2S step is applied on edge (a, d), vertex e and the new vertex is f. Therefore $LCA_T(a, d, e) = \{a, b, c, d, e\}$, in this case, the root of $\mathbf{T}$. Notice that there is no vertex $u \in LCA_T(a, d, e)$ such that $\{a, d, u\}$ is a triple of hinges for $LCA_T(a, d, e)$. Thus the final graph is not tree-decomposable.

Now consider the Hennberg sequences in Figure 8. In both sequences, the reduction H2S is applied on edge (a, d) and the new vertex is f. In the sequence at the top, the third vertex in $V(G)$ considered is c and $LCA_T(a, d, c) = \{a, b, c, d\}$. Notice that $\{a, d, c\}$ is a triple of hinges for $LCA_T(a, d, c)$. In the sequence at the bottom, the third vertex considered is b. Again $LCA_T(a, d, b) = \{a, b, c, d\}$ and $\{a, d, c\}$ is a triple of hinges for $LCA_T(a, d, b)$. Consequently both sequences create tree-decomposable graphs.

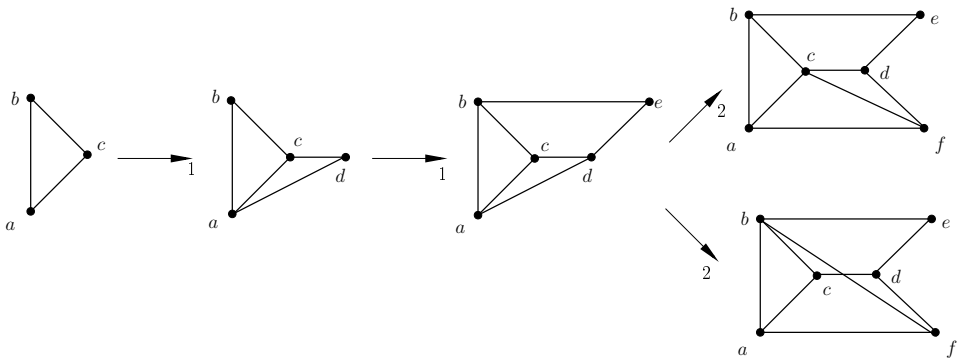

Figure 8: Two Henneberg sequences in H2S that create a well constrained tree-decomposable graph.

Corollary 3.6 Let $G = (V, E)$ be a tree-decomposable graph and (v_1, v_2) be an arbitrary edge in $E(G)$. Then there is always a vertex $u \in V(G)$ different from v_1 and v_2 such that the derivation $G \rightarrow_2 G^*$ involving the edge (v_1, v_2) and vertex u creates a tree-decomposable graph G^*.

This result means that any Henneberg derivation $K_3 \rightarrow^*_{\mathbf{HS}} G$ where G is tree decomposable can be extended to build a tree-decomposable graph with an arbitrary order.

4 Algorithms

In this section we describe two algorithms. The first one builds tree-decomposable graphs of an arbitrary order using Henneberg sequences. The second one updates the tree-decomposition associated with the graph under construction as the Henneberg sequence progresses.

4.1 Creating Tree-decomposable Henneberg Graphs

The algorithm to build tree-decomposable graphs computes the required graph $G = (V, E)$ as the Henneberg sequence $K_3 \rightarrow^*_{\mathbf{HS}} G$ where reductions in $\rightarrow_{\mathbf{HS}}$ are applied according to Theorem 3.5. Besides having the freedom to fix an arbitrary graph order, we are interested in generating graphs within the $\mathbf{H}_2$ family which are more general than graphs in $\mathbf{H}_1$. Procedures are described in Algorithms 1 to 4.

Algorithm 1 Building a tree-decomposable graph of a given order

```
procedure generate_Graph(max_order)
    G = K3
    T = Tree-decomposition of K3
    while G.order < max-order do
        HS = select_Step_Type()
        if HS == H1S then
            add_H1S(G)
            updateT_H1S(T)
        else
            add_H2S(G, T)
            updateT_H2S(T)
    return G
```

Algorithm 2 Add a H1S to a tree-decomposable graph

procedure add_H1S(G)
v1 = Select a vertex from V(G)
v2 = Select a vertex from V(G) different from v1
v = New vertex
V(G) = V(G) ∪ {v}
E(G) = E(G) ∪ {(v, v1), (v, v2)}
return G

Algorithm 3 Add a H2S to a tree-decomposable graph

procedure add_H2S(G, T)
e = Select an edge from E
v1 = e.v1
v2 = e.v2
v3 = identify_Vertex_V3(G, T, v1, v2)
v = new_vertex
V(G) = V(G) ∪ {v}
E(G) = (E(G)- {(v1, v2)}) ∪ {(v, v1), (v, v2), (v, v3)}
return G

Algorithm 4 Compute a third vertex for the H2S

procedure identify_Vertex_V3(G,T,v1,v2)
L = LCAT(T, v1, v2)
Let G1, G2, G3 be the tree-decomposition of L
Let v1 ∈ G1 and v2 ∈ G2
v3 = Select a vertex from G1 or from G2 different from v1 and v2
return v3

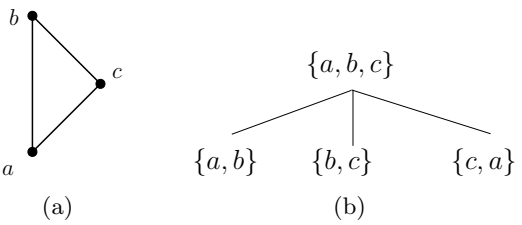

Figure 9: a) Triangle. b) Triangle tree-decomposition.

4.2 Updating the Tree Decomposition

We consider tree-decomposable graphs created by Henneberg sequences with the triangle K_3 as the starting term, that is, $K_3 \rightarrow^*_{\mathbf{HS}} G$. Notice that the tree decomposition of K_3, shown in Figure 9, is trivial.

As the Henneberg sequence evolves creating new and larger graphs, when a H2S reduction is applied, the graph tree decomposition plays a central role to identify candidate vertices on which the reduction generates a new tree-decomposable graph. Thus it is convenient to keep the tree decomposition properly updated. Depending on the kind of reduction applied, there are two different cases. First assume that a H1S reduction is applied, $G \rightarrow_1 G^*$, over the vertices v_1, v_2 and that the new vertex is u. To facilitate the implementation we distinguish two different situations. In the first one, vertices v_1, v_2 bound an edge which is a leaf node in $\mathbf{T}$. That is, in the tree decomposition there is a node $\hat{G}$ decomposed into $\hat{G}_1, \hat{G}_2$ and $(\{v_1, v_2\}, \{(v_1, v_2)\})$ as depicted in Figure 10a. $\mathbf{T}$ is updated in two steps, see Figure 10b,

1. Replace the leaf node graph $(\{v_1, v_2\}, \{(v_1, v_2)\})$ with a tree the root of which is the graph $\hat{G}_3 = (\{v_1, v_2, u\}, \{(v_1, v_2), (u, v_1), (u, v_2)\})$ and tree-decomposition $\hat{G}'_1 = (\{v_1, v_2\}, \{(v_1, v_2)\})$, $\hat{G}'_2 = (\{u, v_1\}, \{(u, v_1)\})$ and, $\hat{G}'_3 = (\{u, v_2\}, \{(u, v_2)\})$.

2. Propagate vertex u and edges $(u, v_1), (u, v_2)$ through $\mathbf{T}$ up to the root.

In the second situation vertices v_1, v_2 belong to different branches of the subtree of $\mathbf{T}$ rooted at $LCA_T(v_1, v_2)$, that is, they bound edges in different tree leaves. The update now takes five steps. See Figure 11.

1. Compute $\hat{G} = LCA_T(v_1, v_2)$ and let $\hat{G}_1$, $\hat{G}_2$, $\hat{G}_3$ be the decomposition of $\hat{G}$ with, say, $v_1 \in V(\hat{G}_i)$ and $v_2 \in V(\hat{G}_j)$ and $i \neq j$.

2. Replace $\hat{G}_1$ with a tree rooted at $\hat{G}_1 \cup \hat{G}_2 \cup \hat{G}_3$ the decomposition of which is $\hat{G}_1$, $\hat{G}_2$, $\hat{G}_3$.

3. Replace $\hat{G}_2$ with $(\{u, v_1\}, \{(u, v_1)\})$.

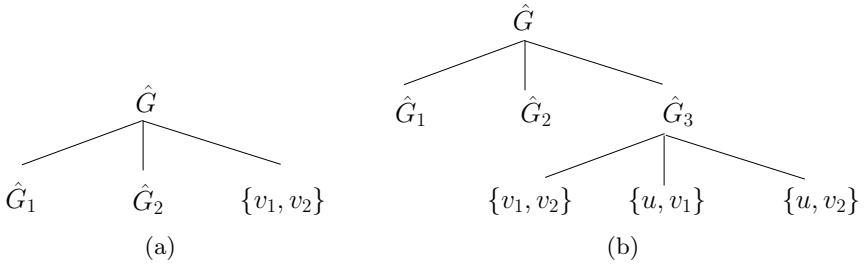

Figure 10: a) Vertices v_1, v_2 bound the edge (v_1, v_2) in a graph leaf node of $\mathbf{T}$. b) Updated tree-decomposition.

4. Replace $\hat{G}_3$ with $(\{u, v_2\}, \{(u, v_2)\})$.
5. Propagate vertex u and edges $(u, v_1), (u, v_2)$ through $\mathbf{T}$ up to the root.

Now assume that the reduction to be applied is the H2S, $G \rightarrow_2 G^*$ where the edge and vertex involed are (v_1, v_2) and v_3 respectively which necessarily belong to different branches in the tree-decomposition $\mathbf{T}$. The new vertex is u. The algorithm has de following steps

1. Compute $\hat{G} = LCA_T(v_1, v_2, v_3)$ and let $\hat{G}_1$, $\hat{G}_2$, $\hat{G}_3$ be the decomposition of $\hat{G}$ with, say, $(v_1, v_2) \in E(\hat{G}_1)$ and $v_3 \in V(\hat{G}_2)$.
2. Replace $\hat{G}_1$ with $(\{u, v_i\}, \{(u, v_i)\})$ where v_i is either v_1 or v_2.
3. Replace $\hat{G}_2$ with the new graph $\hat{G}'_2$ such that $V(\hat{G}'_2) = V(\hat{G}_2) \cup \{u, v_j\}$ and $E(\hat{G}'_2) = (E(\hat{G}_1) \cup E(\hat{G}_2)) - \{(v_1, v_2)\}) \cup \{(u, v_j), (u, v_3)\}$, where v_j is either v_1 or v_2 and $v_j \neq v_i$.
4. Propagate vertex u and edges $(u, v_1), (u, v_2), (u, v_3)$ through $\mathbf{T}$ up to the root.

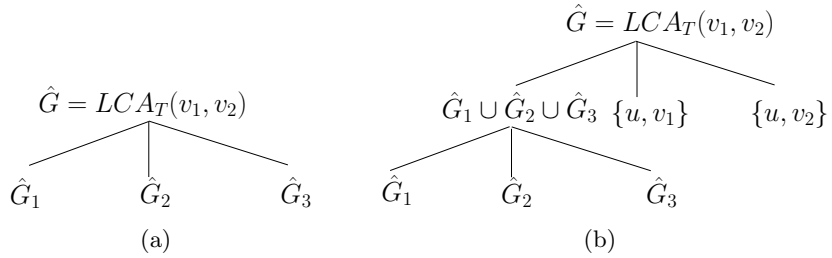

Figure 11: a) Vertices v_1 and v_2 belong to different branches of the subtree of $\mathbf{T}$ rooted at $LCA_T(v_1, v_2)$. b) Updated tree-decomposition.

5 Conclusion

The goal of geometric constraint problem solving is to describe how to place geometric elements with respect to each other in such a way that the constraints are fulfilled. Moreover, actual realizations of the resulting geometric construction should be rigid objects.

To accomplish this goal, the problem must be well constrained, that is, the number of solutions to the problem must be finite. If a number of constraints are missing, the geometric realization is no longer rigid and there are an infinity of solutions to the problem. The most interesting way to limit the number of solutions is to transform the under-constrained problem into a well constrained one by adding as many constraints as needed.

The approach developed here offers a simple tool to automatically generate a set of constraints that defines a well constrained problem over a set of geometric elements using Henneberg reductions in H1S and H2S. Moreover, the approach guarantees that the resulting graph is tree-decomposable and therefore solvable by constructive graph-based DR-solvers if the starting problem is abstracted as K_3. The approach clearly applies as far as the graph with missing constraints used as starting graph in the Henneberg sequence is tree-decomposable. No upper limit is imposed on the number of geometric objects in the problem. Because of generality in the representation, we have fixed the triangle as the starting graph for Henneberg sequences. However, the approach is trivially extended to consider as the starting graph an edge $G = (\{a, b\}, \{(a, b)\})$ by requiring that the first Henneberg step in the derivation of the desired graph be a H1S construction.

To illustrate the approach, the algorithms described in this work select H1S and H2S Henneberg constructions in a random way. An avenue to explore is to study different strategies to select the type of the next construction step to be applied depending on either technological rules convenient for the specific design at hand or the kind of geometric elements on which the new constraint should be defined. Similarly, in our implementation vertices and edges already included in the graph under construction that are involved in the next construction step are selected at random among all the candidates. Strategies to select vertices and edges, when more than one candidate have been found, would be of great interest. Successful approaches to solve these issues would be of help, for example, in the development of techniques to explore different ways to build up rigid frameworks from smaller ones in engineering.

References

[1] Ciprian Borcea and Ileana Streinu. The number of embeddings of minimally rigid graphs. *Discrete & Computational Geometry*, 31:287–303, 2004.

[2] András Frank and László Szegö. An extension of a theorem of Henneberg and Laman. Technical Report TR-2001-05, Egérvary Research Group on Combinatorial Optimization, www.cs.elte.hu/egres, February 2001.

[3] I. Fudos and C.M. Hoffmann. Constraint-based parametric conics for CAD. *Computer Aided Design*, 28(2):91–100, 1996.

[4] I. Fudos and C.M. Hoffmann. A graph-constructive approach to solving systems of geometric constraints. *ACM Transactions on Graphics*, 16(2):179–216, April 1997.

[5] L. Henneberg. *Die Graphische Statik der Starren Systeme.* Leipzig, 1911. Johnson Reprint 1968.

[6] M. R. Hidalgo. *Geometric Constraint Solving in a Dynamic Geometry Framework.* PhD thesis, Universitat Politècnica de Catalunya, 2013.

[7] C.M. Hoffmann and R. Joan-Arinyo. Symbolic constraints in constructive geometric constraint solving. *Journal of Symbolic Computation*, 23:287–300, 1997.

[8] C.M. Hoffmann and R. Joan-Arinyo. A brief on constraint solving. *Computer-Aided Design and Applications*, 2(5):655–663, 2005.

[9] C.M. Hoffmann, A. Lomonosov, and M. Sitharam. Decompostion Plans for Geometric Constraint Problems, Part II: New Algorithms. *Journal of Symbolic Computation*, 31:409–427, 2001.

[10] C.M. Hoffmann, A. Lomonosov, and M. Sitharam. Decompostion Plans for Geometric Constraint Systems, Part I: Performance Measurements for CAD. *Journal of Symbolic Computation*, 31:367–408, 2001.

[11] Bill Jackson and Tibor Jordan. Globally rigid circuits of the direction–length rigidity matroid. *Journal of Combinatorial Theory, Series B*, 100(1):1–22, january 2010.

[12] C. Jerman, G. Trombettoni, B. Neveu, and P. Mathis. Decomposition of geometric constraint systems: A survey. *International Journal of Computational Geometry and Applications*, 16(5-6):379–414, 2006.

[13] R. Joan-Arinyo and A. Soto. A correct rule-based geometric constraint solver. *Computer & Graphics*, 21(5):599–609, 1997.

[14] R. Joan-Arinyo and A. Soto-Riera. Combining constructive and equational geometric constraint solving techniques. *ACM Transactions on Graphics*, 18(1):35–55, January 1999.

[15] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana. On the domain of constructive geometric constraint solving techniques. In R. Duricovic and S. Czanner, editors, *Spring Conference on Computer Graphics*, pages 49–54, Budmerice, Slovakia, April 25-28 2001. IEEE Computer Society, Los Alamitos, CA.

[16] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana. On the completion of underconstrained geometric constraint problems. In *Third International NAISO Symposium on Engineering of Intelligent Systems*, Málaga, Spain, September 24-27 2002.

[17] R. Joan-Arinyo, A. Soto-Riera, S. Vila-Marta, and J. Vilaplana. Transforming an underconstrained geometric constraint problem into a well-constrained one. In G. Elber and V.Shapiro, editors, *Eight Symposium on Solid Modeling and Applications*, pages 33–44, Seattle (WA) USA, June 16-20 2003. ACM Press.

[18] J.W. Klop. Term rewriting systems. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Background: Computational Structures*, volume 2 of *Handbook of Logic in Computer Science*, pages 1–117. Clarendon Press, 1992.

[19] G. Laman. On graphs and rigidity of plane skeletal structures. *Journal of Engineering Mathematics*, 4(4):331–340, October 1970.

[20] N. Mata. Solving incidence and tangency constraints in 2D. Technical Report LSI-97-3R, Department LiSI, Universitat Politècnica de Catalunya, 1997.

[21] B. Servatius, H. Servatius, and J. Graver. *Combinatorial Rigidity*. J. E. Humphreys, R. C. Kirby, L. W. Small, 1993.

[22] Brigitte Servatius and Herman Servatius. Rigidity, global rigidity, and graph decomposition. *European Journal of Combinatorics*, 31(4):1121–1135, 2010.

[23] A. Soto. *Satisfacció de restriccions geomètriques en 2D*. PhD thesis, Universitat Politècnica de Catalunya, Dept. Llenguatges i Sistemes Informàtics, 1998. (Written in Catalan).

[24] Tiong-Seng Tay and Walter Whiteley. Generating isostatic frameworks. *Structural Topology*, (11):21–69, 1985.

A Proof of Theorems

In this section we develop proofs for the claims in the manuscript. Recall from Section 3 that $\rightarrow_1$ and $\rightarrow_2$ denote respectively the H1S and the H2S constructions and that $\rightarrow_{\mathbf{HS}}$ denotes de set including both construction steps. An arbitrary sequence of reduccions in $\rightarrow_{\mathbf{HS}}$ is denoted by $\rightarrow^*_{\mathbf{HS}}$. When the sequence includes just one of the two rules in $\rightarrow_{\mathbf{HS}}$, it will be denoted as either $\rightarrow^*_1$ or $\rightarrow^*_2$.

In what follows, we will be interested in distinguishing reductions depending on the geometric elements involved. A H1S reduction which adds the new vertex u and two edges (u, v_1) and (u, v_2) will be denoted as $\longrightarrow_{1,\{v_1,v_2,u\}}$. A H2S reduction which adds the new vertex u, removes edge (v_1, v_2) and adds three new edges (u, v_1), (u, v_2) and (u, v_3) will be denoted as $\longrightarrow_{2,\{v_1,v_2,v_3,u\}}$.

Theorem 3.1 Let G and G^* be two graphs such that $G \rightarrow_1 G^*$. Then G^* is tree-decomposable if and only if G is tree-decomposable.

Proof
Assume that G is a tree-decomposable graph and consider the reduction $G \rightarrow_{1,\{v_1,v_2,u\}} G^*$. Then

$$G^* = (V(G) \cup \{u\},\ E(G) \cup \{(v_1, u), (v_2, u)\})$$

and the graphs G, $G_1 = (\{v_1, u\}, \{(v_1, u)\})$ and $G_2 = (\{v_2, u\}, \{(v_2, u)\})$ define a tree-decomposition for G^* with G tree-decomposable. Assume now that G^* is tree-decomposable. By the same argument, G is tree-decomposable. □

Theorem 3.2 Let $G = (V, E)$ be a graph in the $\mathbf{H}_1$ family and G^* be the graph such that $G \rightarrow_2 G^*$, where the H2S step involves edge $(v_1, v_2) \in E(G)$ and vertex $u \in V(G)$. If the triple $\{v_1, v_2, u\}$ are hinges for a reduction in the derivation $K_3 \rightarrow^*_1 G$, then $G^* \in \mathbf{H}_1$.

Proof
Assume that $G \in \mathbf{H}_1$, then the derivation $K_3 \rightarrow^*_1 G$ is a Henneberg sequence for G. If we assume that $\{v_1, v_2, u\}$ is a triple on which a H1S construction has been applied, the derivation for G can be rewritten in general as

$$K_3 \rightarrow^*_1 G' \rightarrow_{1,\{v_1,v_2,u\}} G'' \rightarrow^*_1 G \tag{1}$$

By definition, reduction $\rightarrow_1$ does not remove graph edges and always connects a new vertex to the bounds of a single edge in $E(G')$ with two new edges. Thus after applying reductions $\rightarrow_{1,\{v_1,v_2,u\}}$ and $\rightarrow^*_1$, edges (v_1, v_2), (u, v_1) and (u, v_2) are in $E(G)$ and clearly v_1, v_2, u are in $V(G)$. Consider the derivation

$$K_3 \rightarrow^*_1 G' \rightarrow_{1,\{v_1,v_2,u\}} G'' \rightarrow^*_1 G \rightarrow_{2,\{v_1,u,v_2,w\}} G^* \tag{2}$$

built by extending the derivation (1) with the reduction $\rightarrow_{2,\{v_1,u,v_2,w\}}$ in H2S. Then

$$V(G^*) = V(G) \cup \{w\}$$

with $v_1, v_2, u \in V(G)$ and

$$E(G^*) = (E(G) - \{(u, v1)\}) \cup \{(w, v_1), (w, u), (w, v_2)\} \tag{3}$$

Reductions in derivation (1) belong to the H1S class and vertices $v_1, v_2, u \in V(G)$ and edges $(v_1, v_2), (u, v_1), (u, v_2) \in E(G)$. Hence the derivation

$$K_3 \rightarrow_1^* G' \rightarrow_{1,\{v_1,v_2,u\}} G'' \rightarrow_1^* G \rightarrow_{1,\{v_2,u,w\}} G^{*'} \tag{4}$$

is well defined. Then

$$V(G^{*'}) = V(G) \cup \{w\}$$

with $v_1, v_2, u \in V(G)$ and

$$E(G^{*'}) = E(G) \cup \{(w, v_2), (w, u)\} \tag{5}$$

Let E' denote the set of edges added to $E(G)$ by $G'' \rightarrow_1^* G$ in the derivation (1). Then

$$E(G) = E(G') \cup \{(u, v_1), (u, v_2)\} \cup E'$$

Replacing $E(G)$ in equation (3) we have (See Figure 12 Top),

$$\begin{aligned} E(G^*) &= (E(G') \cup E' \cup \{(u, v_2)\}) \cup \{(w, v_1), (w, u), (w, v_2)\} \\ &= E(G') \cup E' \cup \{(u, v_2), (w, v_1), (w, u), (w, v_2)\} \end{aligned}$$

Replacing $E(G)$ in equation (5) we have (See Figure 12 Bottom),

$$\begin{aligned} E(G^{*'}) &= (E(G') \cup E' \cup \{(u, v_1), (u, v2)\}) \cup \{(w, v_2), (w, u)\} \\ &= E(G') \cup E' \cup \{(u, v_1), (u, v_2), (w, v_2), (w, u)\} \end{aligned}$$

A proper relabeling of vertices u and w shows that $E(G^*) = E(G^{*'})$. This along with the fact that $V(G^*) = V(G^{*'})$ lead to $G^* = G^{*'}$. Thus graph G^* belongs to $\mathbf{H}_1$ because derivation (4) is in H1S. □

Lemma 3.3 Let $G = (V, E)$ be a tree-decomposable Henneberg graph for which G_1, G_2, G_3 is a decomposition induced by the triple $\{v_1, v_2, x\}$ and such that $G_i = (\{v_1, v_2\}, \{(v_1, v_2)\})$ for some $1 \leq i \leq 3$. Then the graph G^* created by the reduction $G \rightarrow_2 G^*$ involving the edge $(v_1, v_2) \in E(G)$ and vertex $v_3 \in V(G)$ is tree-decomposable.

Proof
Refer to Figure 13. Without loss of generality, assume that the given graph

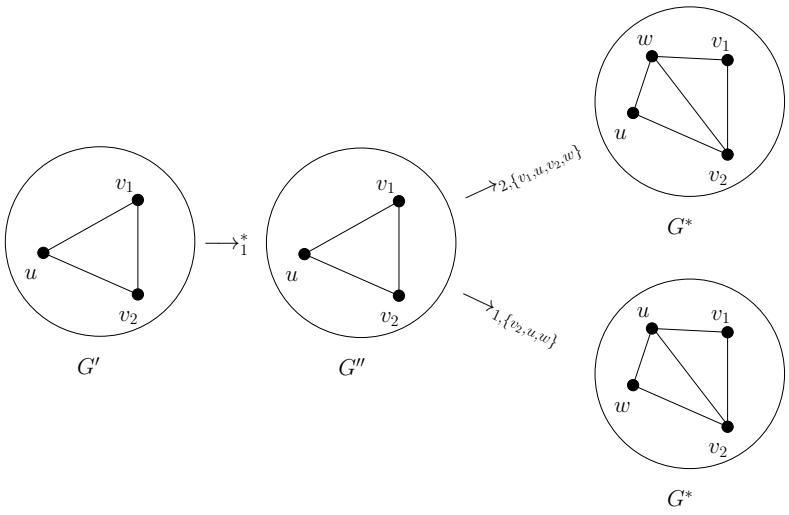

Figure 12: Derivation $\longrightarrow_1^*$ preserves edges. Top) Reduction $\longrightarrow_{2,\{v_1,u,v_2,w\}}$ removes egde (v_1, u) and adds edges (w, v_1), (w, u) and (w, v_2). Bottom) Reduction $\longrightarrow_{1,\{v_2,u,w\}}$ preserves edges plus adds edges (w, v_2) and (w, u).

G is decomposed into G_1, G_2, G_3 with $G_1 = (\{v_1, v_2\}, \{(v_1, v_2)\})$, $v_3, v_1 \in V(G_3)$ and $v_2 \in V(G_2)$.

The graph G^* created by the reduction $G \rightarrow_{2,\{v_1,v_2,v_3,u\}} G^*$ is such that $V(G^*) = V(G) \cup \{u\}$ and $E(G^*) = (E(G) - \{(v_1, v_2)\}) \cup \{(u, v_1), (u, v_2), (u, v_3)\}$. Then G^* can be decomposed into three subgraphs, $G_1^* = (\{u, v_2\}, \{(u, v_2)\})$, $G_2^* = G_2$ and, $G_3^* = (V(G_3) \cup \{u\}, E(G_3) \cup \{(u, v_1), (u, v_3)\})$. Subgraph G_1^* is a leaf node in a tree-decomposition. Subgraph G_2^* is clearly tree-decomposable. The triple $\{u, v_1, v_3\}$ decomposes the graph G_3^* into G_3, $(\{u, v_1\}, \{(u, v_1)\})$ and $(\{u, v_3\}, \{(u, v_3)\})$. The fact that, by hypothesis, G_3 is tree-decomposable completes the proof. □

Lemma 3.4 Let $G = (V, E)$ be a tree-decomposable Henneberg graph for which G_1, G_2, G_3 is a decomposition induced by the triple $\{v_1, v_2, x\}$ and such that edge $(v_1, v_2) \in E(G_i)$ for some $1 \leq i \leq 3$. If there is a vertex $w \in V(G_i)$ such that the triple $\{v_1, v_2, w\}$ decomposes G_i, then the graph G^* created by the reduction $G \rightarrow_2 G^*$ involving edge $(v_1, v_2) \in E(G)$ and a third vertex $v_3 \in V(G)$ is tree-decomposable.

Proof
Let G_1, G_2, G_3 be the decomposition induced by the triple $\{v_1, v_2, x\} \in V(G)$ in the tree-decomposable graph G, as depicted in Figure 14a. Without loss of generality, assume that egde $(v_1, v_2) \in E(G_1)$ and that $V(G_1) \cap V(G_2) = \{v_2\}$, $V(G_2) \cap V(G_3) = \{x\}$ and $V(G_3) \cap V(G_1) = \{v_1\}$. By hypothesis G is tree-decomposable, hence G_1 is also tree-decomposable. In particular, assume that there is a vertex $w \in V(G_1)$ such that the triple $\{v_1, v_2, w\}$

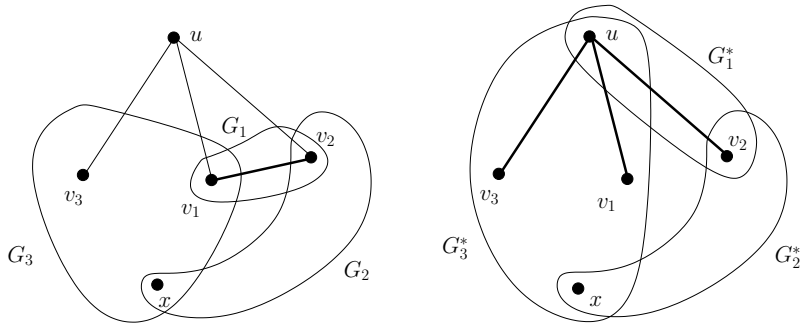

Figure 13: a) The given graph $G = G_1 \cup G_2 \cup G_3$ with $G_1 = (\{v_1, v_2\}, \{(v_1, v_2)\})$. b) Decomposition for the graph G^* created by reduction $G \rightarrow_{2,\{v_1,v_2,v_3,u\}} G^*$.

decomposes G_1 into G_1', G_2', G_3' with, say, $G_1' = (\{v_1, v_2\}, \{(v_1, v_2)\})$. See Figure 14a. Now consider the reduction $G \rightarrow_{2,\{v_1,v_2,v_3,u\}} G^*$ and, in G^*, define the graphs

$$G_1^* = (V(G_2) \cup V(G_3) \cup \{u\}, E(G_2) \cup E(G_3) \cup \{(u, v_1), (u, v_2), (u, v_3)\})$$

$$G_2^* = G_2', \qquad G_3^* = G_3'$$

Clearly, G_1^*, G_2^*, G_3^* is a decomposition induced in G^* by the triple $\{v_1, v_2, w\}$. G_2^* and G_3^* are tree-decomposable because G_2' and G_3' are tree-decomposable. Consider the graph G_1^* as the graph created by a H2S construction on edge (v_1, v_2) and vertex v_3 on the graph $G_1 \cup G_2$. See Figure 14b. Apply Lemma 3.3 to show that G_1^* is tree-decomposable. Thus G^* is tree-decomposable. □

Theorem 3.5 Let $G = (V, E)$ be a tree-decomposable Henneberg graph with $\mathbf{T}$ the associated tree-decomposition. Let G^* be the grah created by the derivation $K_3 \rightarrow^*_{\mathbf{HS}} G' \rightarrow^*_{\mathbf{HS}} G \rightarrow_2 G^*$ where reduction $\rightarrow_2$ is applied on edge $(v_1, v_2) \in E(G') \subset E(G)$ and vertex $v_3 \in V(G') \subset V(G)$. Let $\hat{G}_1, \hat{G}_2, \hat{G}_3$ be a decomposition of $LCA_T(v_1, v_2, v_3)$ with $(v_1, v_2) \in E(\hat{G}_i)$ for some $1 \leq i \leq 3$. Then G^* is tree-decomposable if either $\hat{G}_i = (\{v_1, v_2\}, \{(v_1, v_2)\})$ or there is a vertex $w \in V(\hat{G}_i)$ such that $\{v_1, v_2, w\}$ is a triple of hinges for $\hat{G}_i$.

Proof
Consider a tree-decomposable graph $G = (V, E)$ with $\mathbf{T}$ as the associated tree-decomposition and denote the graph $LCA_T(v_1, v_2, v_3) \subset G$ as $\hat{G}$. $\hat{G}$ is tree-decomposable because so is G. Let $\hat{G}_1$, $\hat{G}_2$, $\hat{G}_3$ denote this decomposition.

First, without loss of generality, assume that $\hat{G}_1 = (\{v_1, v_2\}, \{(v_1, v_2)\})$ and that $v_3 \in V(\hat{G}_3)$, $\hat{G}_1 \cap \hat{G}_2 = \{v_2\}$, $\hat{G}_2 \cap \hat{G}_3 = \{x\}$ and $\hat{G}_3 \cap \hat{G}_1 = \{v_1\}$.

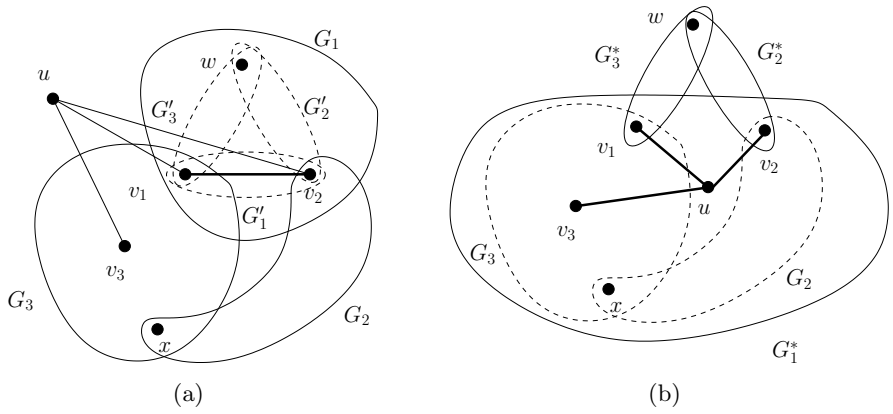

Figure 14: a) Given graph G. The triple $\{v_1, v_2, x\}$ induces the decomposition G_1, G_2, G_3 and $\{v_1, v_2, w\}$ is a triple for G_1. b) Graph G^* created by the reduction $G \rightarrow_{2,\{v_1,v_2,v_3,u\}} G^*$ and decomposition induced in G^* by the triple $\{v_1, v_2, w\}$.

Reduction $\rightarrow_{2,\{v_1,v_2,v_3,u\}}$ in the derivation $K_3 \rightarrow^*_{\mathbf{HS}} G' \rightarrow^*_{\mathbf{HS}} G \rightarrow_2 G^*$ only affects edges and vertices in $\hat{G}$. Hence the reduction $\hat{G} \rightarrow_{2,\{v_1,v_2,v_3,u\}} \hat{G}^*$ is well defined. Moreover, by Lemma 3.3, the graph $\hat{G}^*$ is tree-decomposable by the triple $\{u, v_2, x\}$.

Let $\hat{\mathbf{T}}^*$ be the tree-decomposition associated to the graph $\hat{G}^*$ and let $\mathbf{T}^*$ be the tree-decomposition resulting from replacing in $\mathbf{T}$ the tree rooted at node $LCA_T(v_1, v_2, v_3)$ with $\hat{\mathbf{T}}^*$, as illustrated in Figure 15. Clearly the resulting tree is a tree decomposition for G^*. Therefore G^* is tree-decomposable.

Now, assume that $v_3 \in V(\hat{G}_3)$, $\hat{G}_1 \cap \hat{G}_2 = \{v_2\}$, $\hat{G}_2 \cap \hat{G}_3 = \{x\}$, $\hat{G}_3 \cap \hat{G}_1 = \{v_1\}$, edge (v_1, v_2) is in $E(\hat{G}_1)$ and there is a vertex $w \in V(\hat{G}_1)$ such that $\{v_1, v_2, w\}$ is a triple for $\hat{G}_1$. Lemma 3.4 along with the rational above show that G^* is tree-decomposable. □

Corollary 3.6 Let $G = (V, E)$ be a tree-decomposable graph and (v_1, v_2) be an arbitrary edge in $E(G)$. Then there is always a vertex $u \in V(G)$ different from v_1 and v_2 such that the derivation $G \rightarrow_2 G^*$ involving the edge (v_1, v_2) and vertex u creates a tree-decomposable graph G^*.

Proof
Let $G = (V, E)$ be the given graph and $\mathbf{T}$ the associated tree decomposition. Let $\hat{G}$ be the node in $\mathbf{T}$ such that G is decomposed into $\hat{G}_1$, $\hat{G}_2$ and $(\{v_1, v_2\}, \{(v_1, v_2)\})$. Apply Theorem 3.5 to the reduction $G \rightarrow_{2,v_1,v_2,u,w} G^*$ with $u \in \{V(\hat{G}_1) \cup V(\hat{G})_2\}$ and $u \notin \{v_1, v_2\}$. □

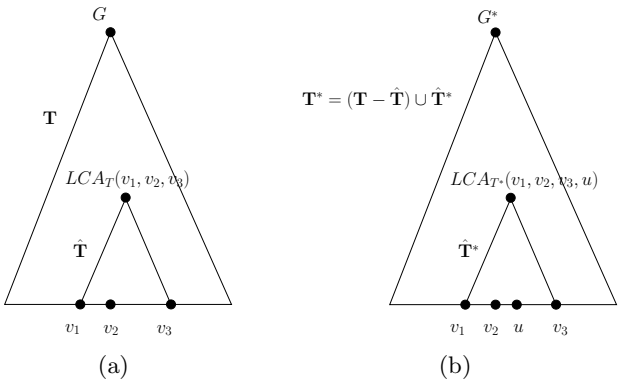

Figure 15: Illustration for Theorem 3.5. a) Tree decomposition $\mathbf{T}$ for the given graph G and tree decomposition $\hat{\mathbf{T}}$ for the subgraph rooted at $LCA_T(v_1, v_2, v_3)$. b) Tree decomposition $\mathbf{T}^*$ resulting after replacing the tree decomposition $\hat{\mathbf{T}}$ in $\mathbf{T}$ with $\hat{\mathbf{T}}^*$. $\mathbf{T}^*$ is a tree decomposition for the graph G^*.